Kosten der Abschirmung von Code und Daten

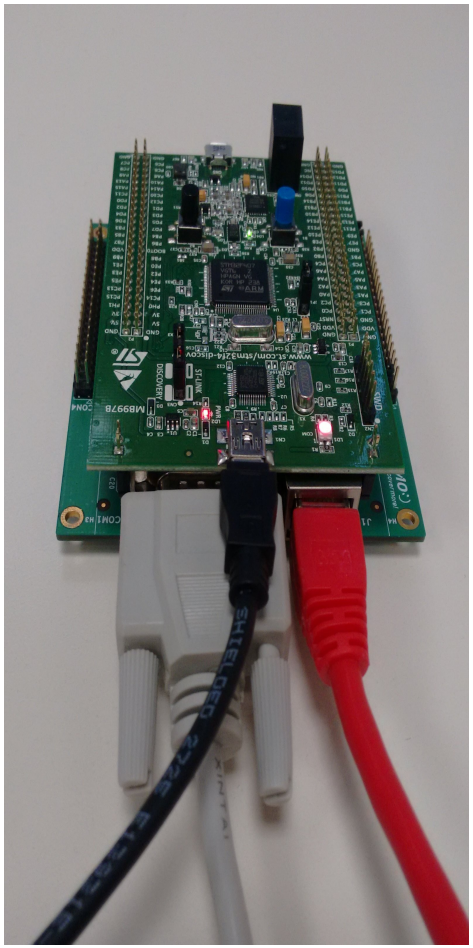
Alexander Züpke, Kai Beckmann, Andreas Zoor, Reinhold Kröger

vorname.nachname@hs-rm.de

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Motivation

Internet der Dinge

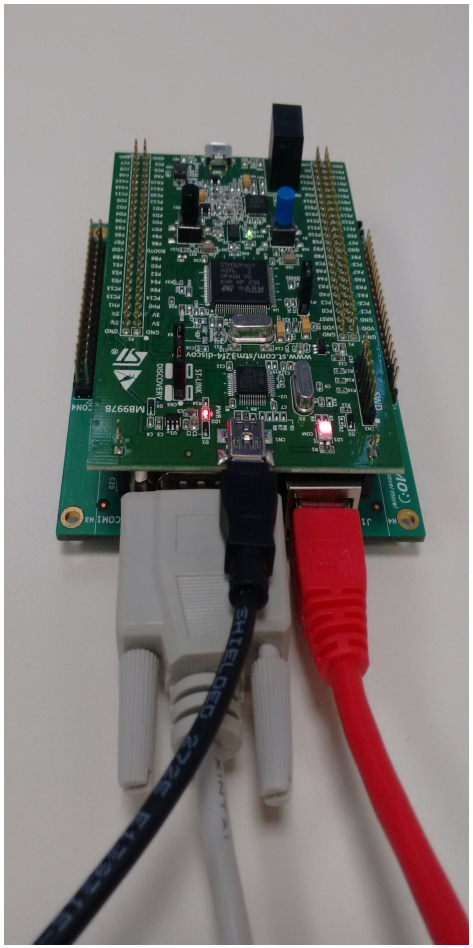


STM32F4 Mikrocontroller

Motivation

Internet der Dinge

Netzwerk-Anschluss!



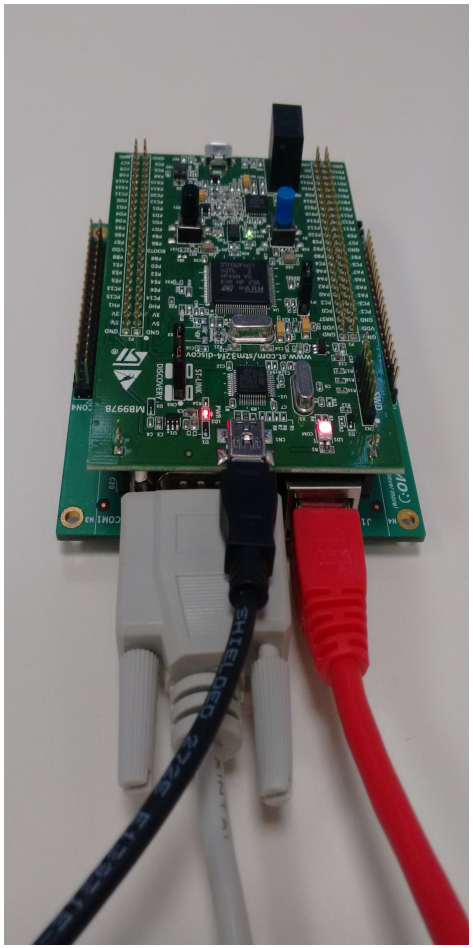
STM32F4 Mikrocontroller

Motivation

Internet der Dinge

Netzwerk-Anschluss!

Angriffe über das Internet?



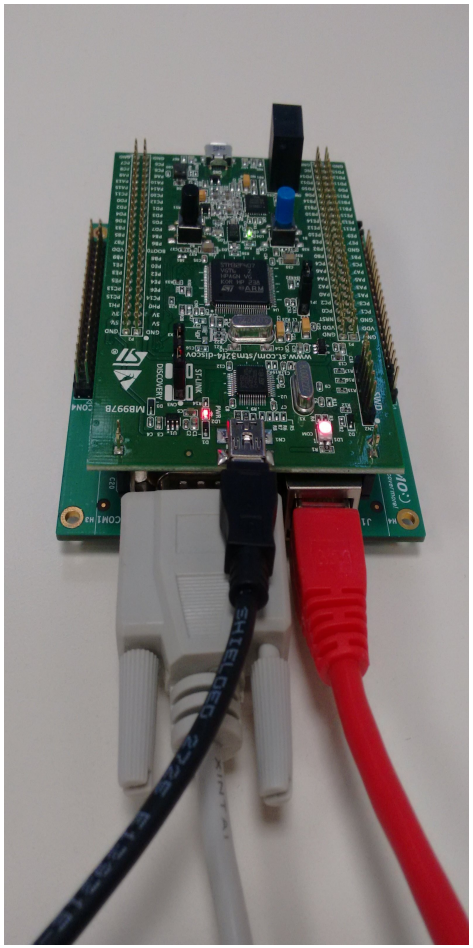
STM32F4 Mikrocontroller

Motivation

Internet der Dinge

Netzwerk-Anschluss!

Angriffe über das Internet?



STM32F4 Mikrocontroller

Abschirmung

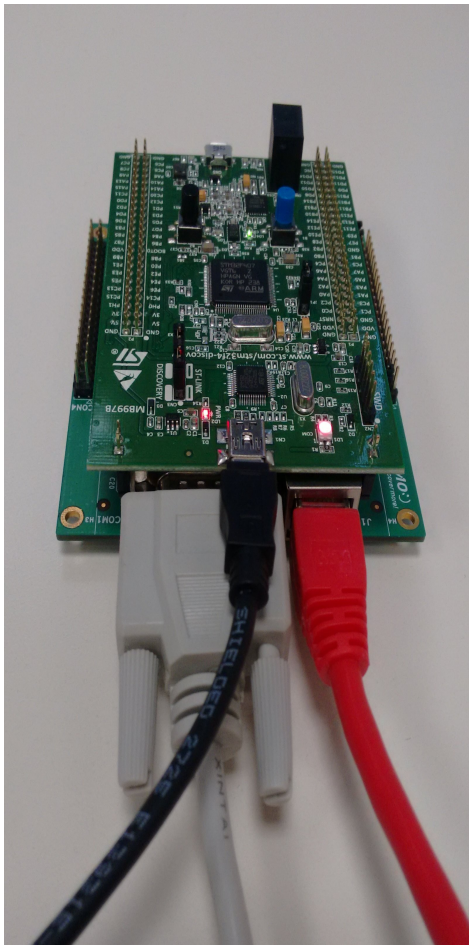
Isolation von Fehlern!

Motivation

Internet der Dinge

Netzwerk-Anschluss!

Angriffe über das Internet?



STM32F4 Mikrocontroller

Abschirmung

Isolation von Fehlern!

Kosten?

Übersicht

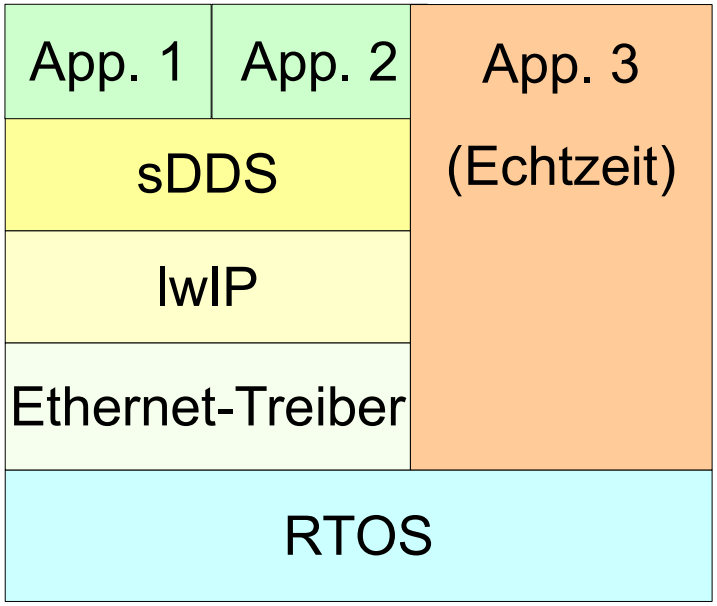
- Software-Architektur
- Trennungskonzepte
- Umsetzung
- Evaluation
- Fazit

Software-Architektur

Software-Architektur

Monolithische IoT-Architektur

ohne Isolation



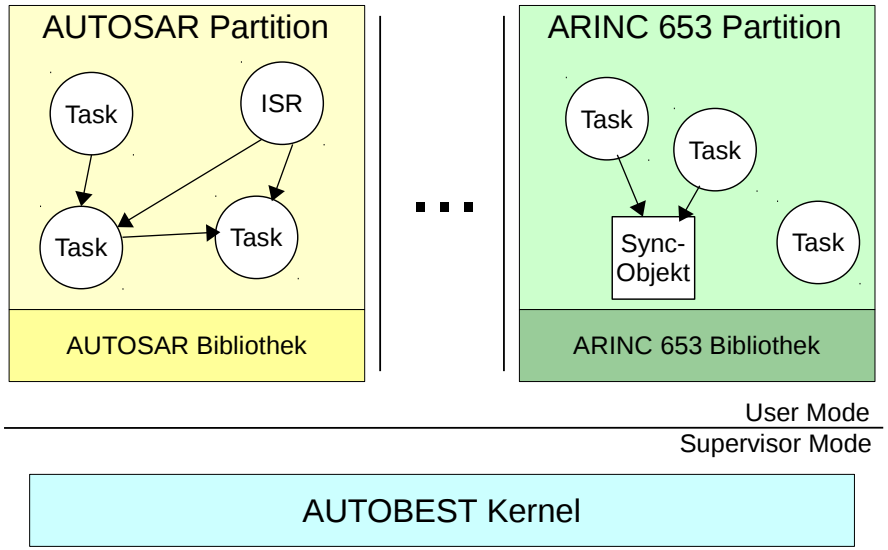
- z.B. FreeRTOS, Contiki, RiotOS

- Fehlende Isolationskonzepte: Alle Software-komponenten teilen den selben Adressraum
- Ein Fehler in der Netzwerkanwendung kann kritische Steuerungsaufgaben kompromittieren

Software-Architektur

AUTOBEST

- Mikrokern für
  MPU-basierte Prozessoren
- Motivation ISO 26262
- Strikte Partitionierung
- Statische Konfiguration
- Kommunikation zwischen Partitionen (“Adressräumen”)
  - Shared-Memory Segmente
  - Synchrone RPC
  - Asynchrone Events

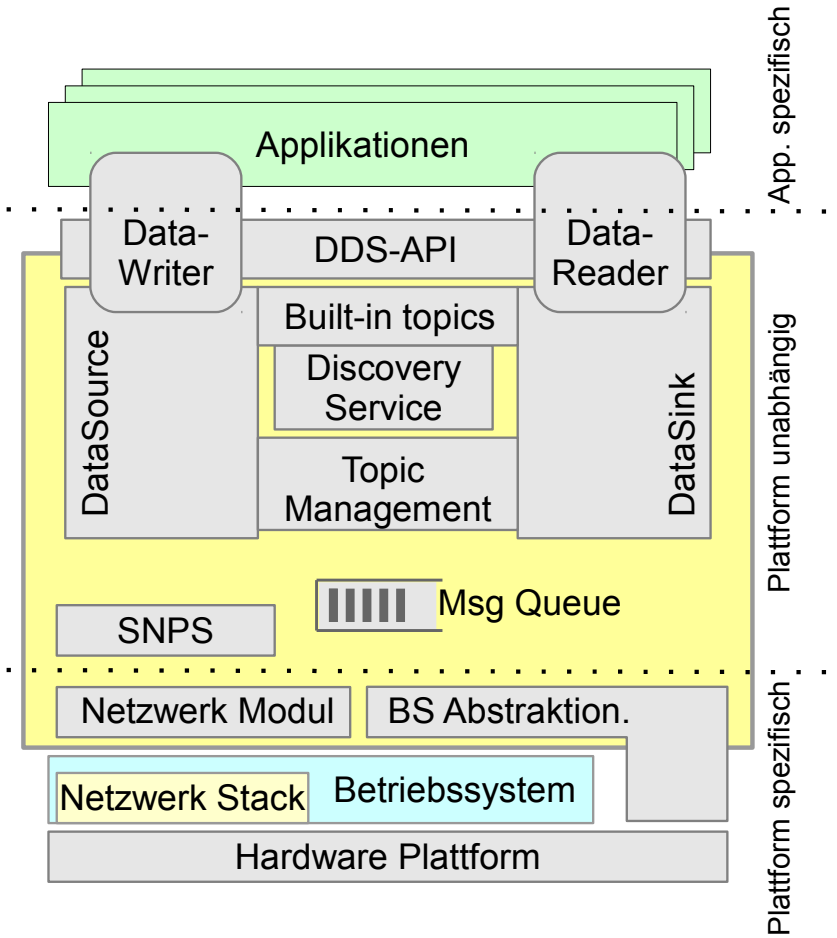


easycore

Software-Architektur

DDS

- Datenzentrierte Middleware mit QoS-Eigenschaften
- OMG Standard
- Publish-Subscribe
- *Topic*: Datenabonnement
- *DataReader* / *DataWriter*



sDDS

- Portable Implementierung für Knoten in Sensornetzwerken
- Kommunikation über UDPv6 Multicast-Gruppen

Software-Architektur

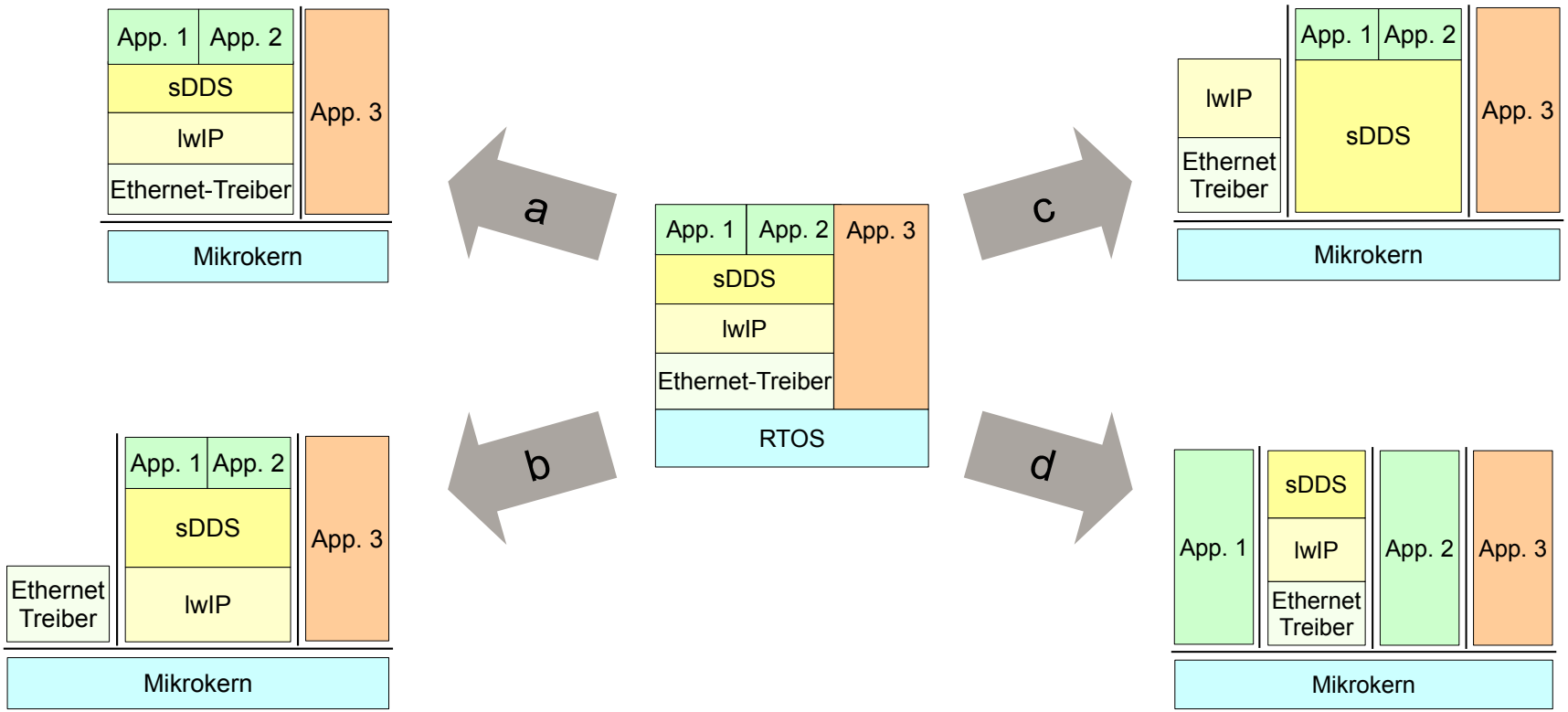
lwIP

- *Lightweight TCP/IP-Stack* für IPv4 und IPv6
- Stark konfigurierbar
- Schnittstelle zum Ethernet-Treiber
  - Austausch von Ethernet-Frames
  - Entkoppelte Pufferstufen
- Schnittstelle zur Anwendung
  - Socket-ähnliches API
  - Blockierendes Senden und Empfangen
- Eigene Speicherverwaltung über sogenannte *Pbufs*

Trennungskonzepte

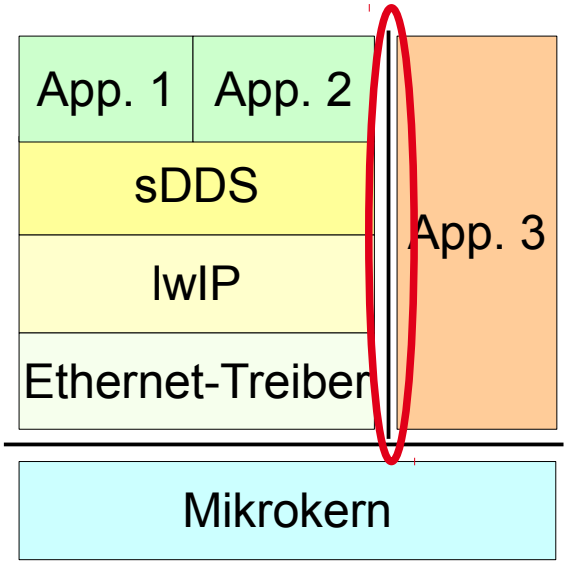
Trennungskonzepte

Isolation der Komponenten in eigene Partitionen



Trennungskonzepte

a. Isolation der Netzwerkkomponenten in eine dedizierte Partition



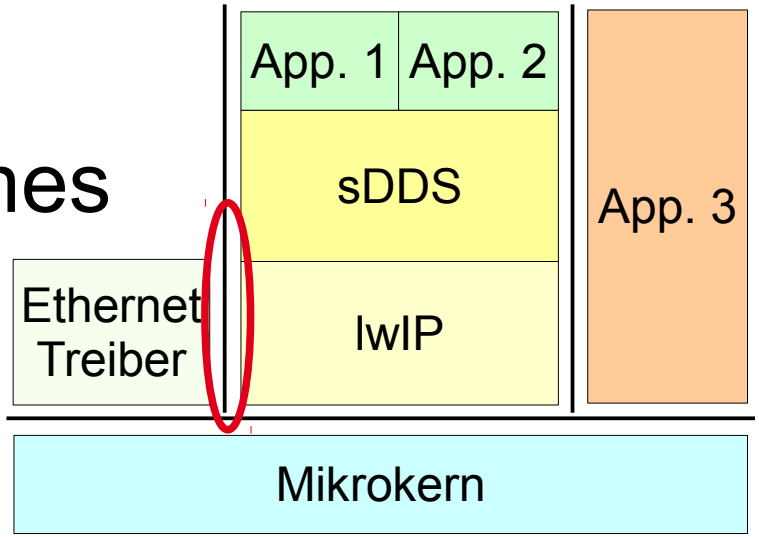
- Ansatz ähnlich Monolith
- Komponentenübergänge über Funktionsaufrufe
- Gute Performance
- Einfache Trennung

→ Baseline für Vergleichmessungen

Trennungskonzepte

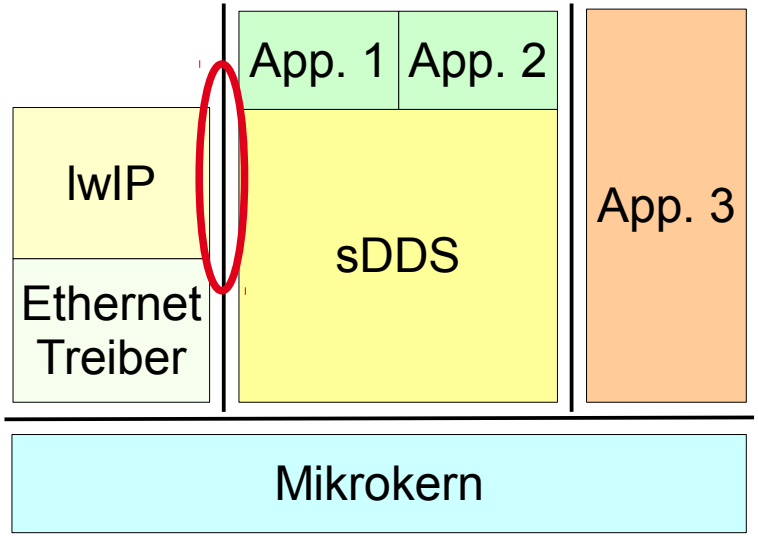
b. Trennung zwischen Ethernet-Treiber und lwIP

- Austausch von Ethernet-Frames
- Auftreten von Bursts
- Entkopplung über Ringpuffer
- Asynchrone Benachrichtigung
- Möglich: DMA-Transfers in Ringpuffern
- Schnittstelle einfach aufzutrennen



Trennungskonzepte

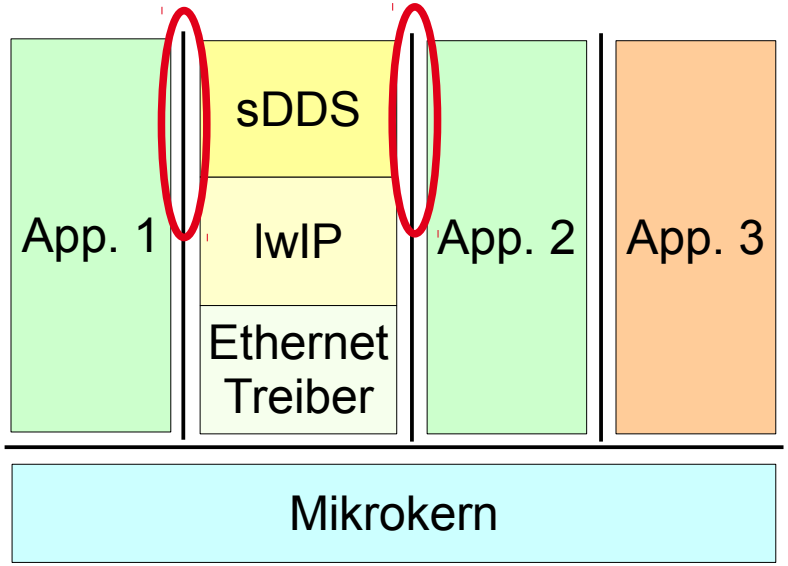
c. Trennung zwischen lwIP und sDDS



- Socket-artige Schnittstelle
- Austausch von UDP-Paketen
- Entkopplung über Ringpuffer
- lwIP meldet empfangene Datagramme asynchron
- sDDS sendet synchron
- Der Aufwand für eine Trennung ist hier höher

Trennungskonzepte

d. Trennung zwischen sDDS und Anwendung



- Komplexe Schnittstelle
  - Benachrichtigungen über Callbacks
  - Blockierendes Senden
  - Das DDS-API erfordert synchrone Aufrufe in den sDDS-Stack
- Schnittstelle ist aufwändig zu trennen, aber die Größe der übertragenen Daten ist i.d.R. klein

Trennungskonzepte

Zusammengefasst:

- Je höher die Schnittstelle angesiedert ist, desto komplizierter wird eine Entkopplung

  → Komplexere Implementierung, mehr Code

- Niedrigere Schnittstellen übertragen mehr Daten

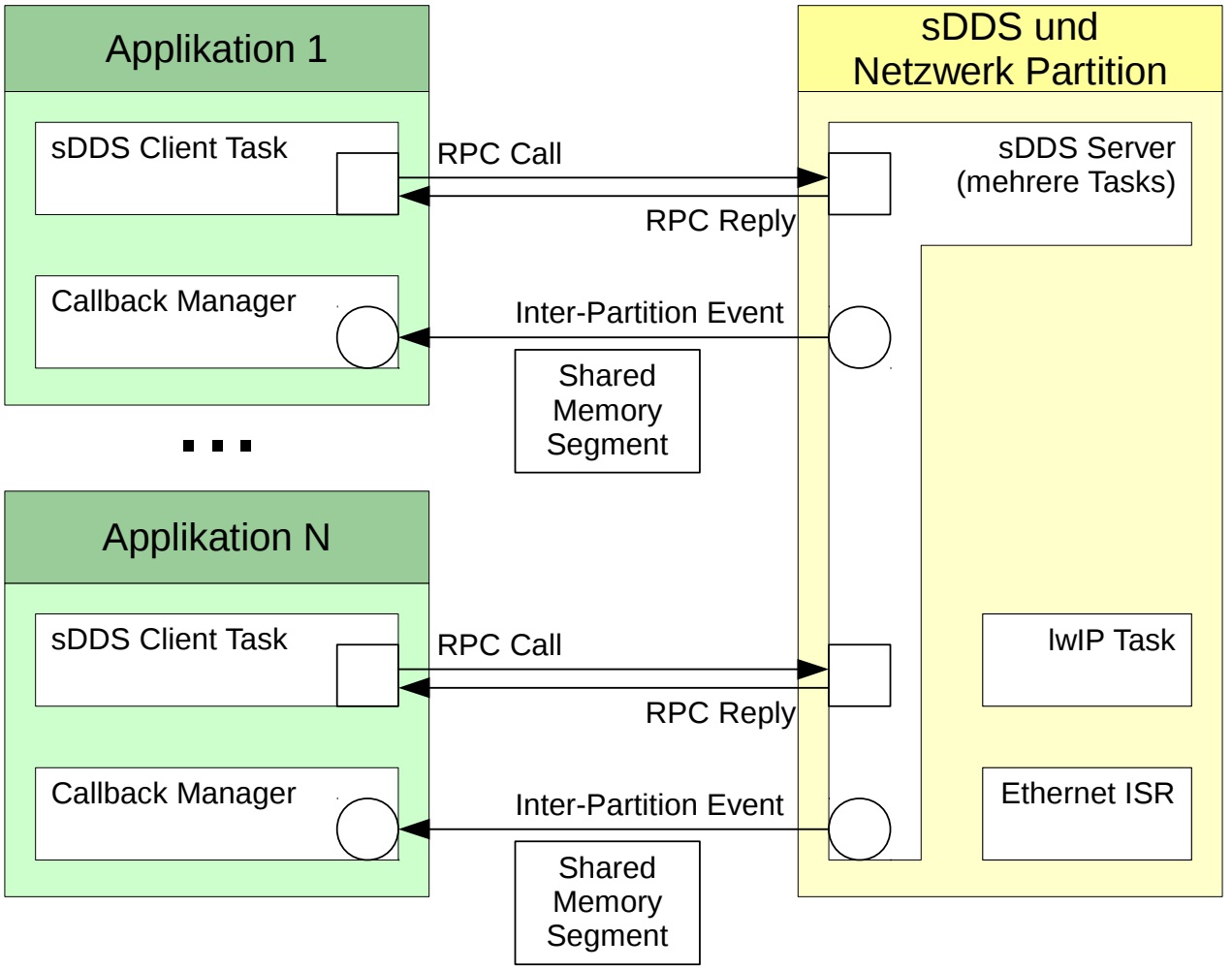
  → höherer Protokoll-Overhead, mehr Daten

- Niedrigere Schnittstellen lassen sich einfacher asynchron entkoppeln
- Höhere Schnittstellen erfordern synchrone Aufrufe

Umsetzung

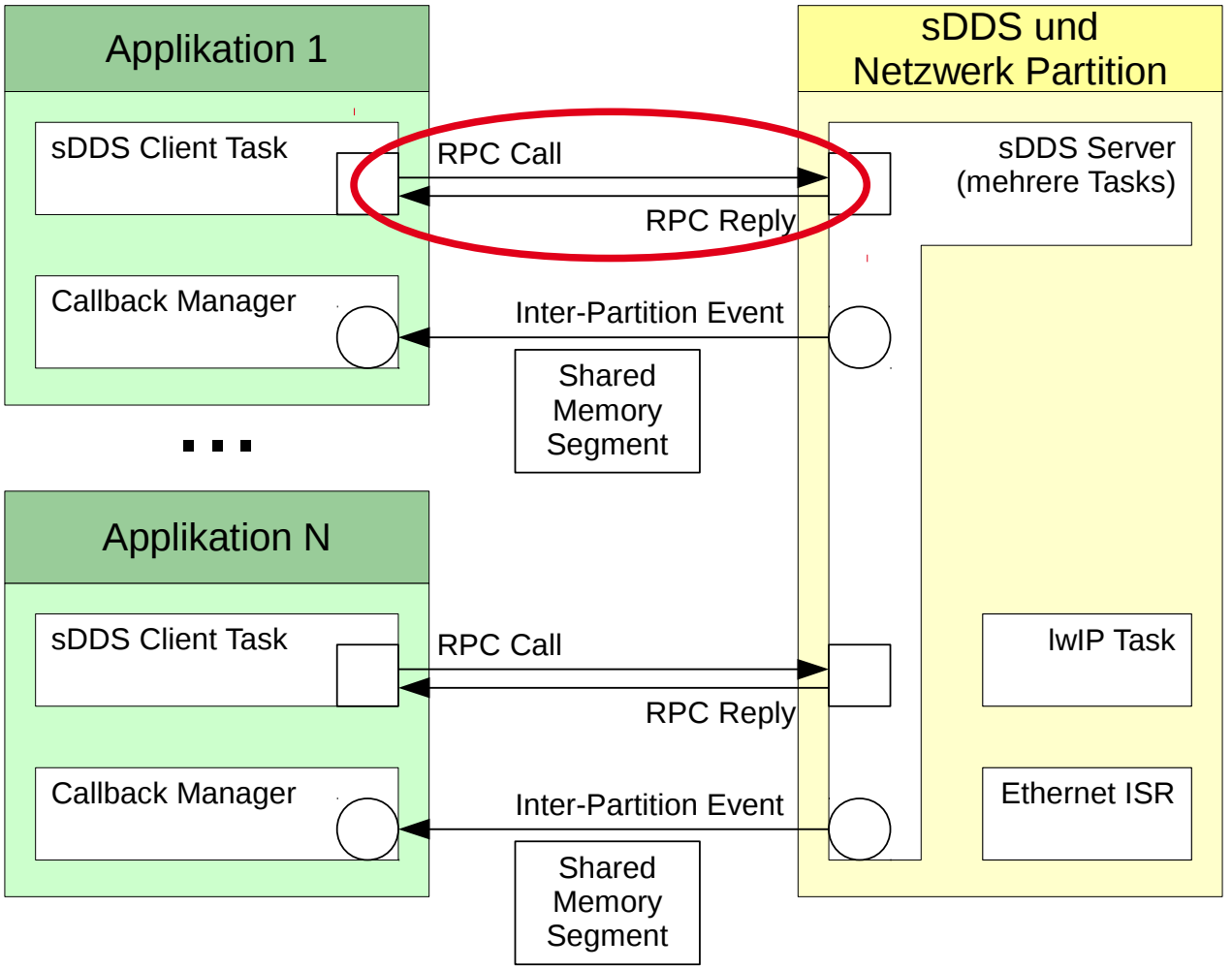
Umsetzung

- Architektur
- DDS API Calls
- Senden
- Empfangen



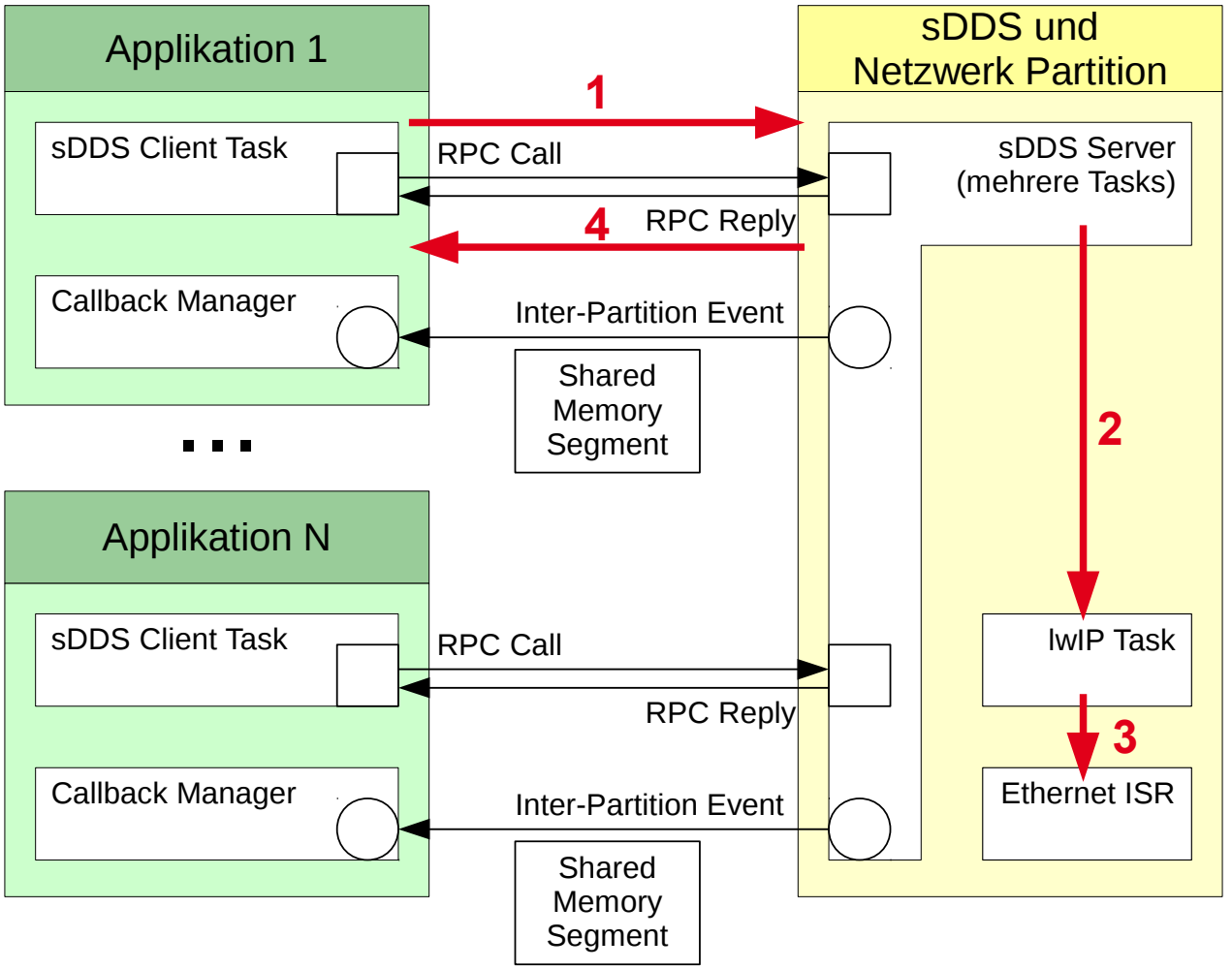
Umsetzung

- Architektur
- DDS API Calls
- Senden
- Empfangen



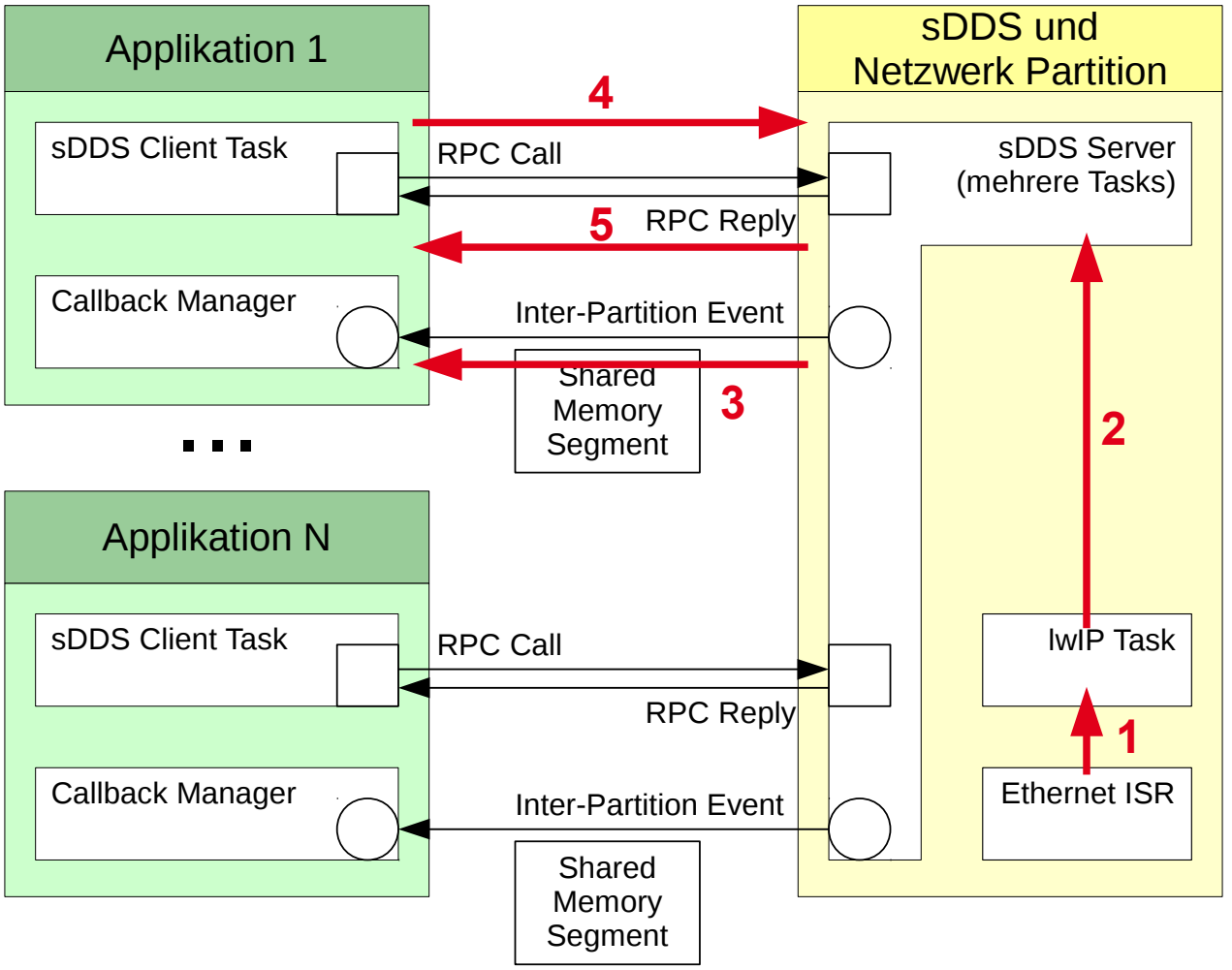
Umsetzung

- Architektur
- DDS API Calls
- Senden
- Empfangen



Umsetzung

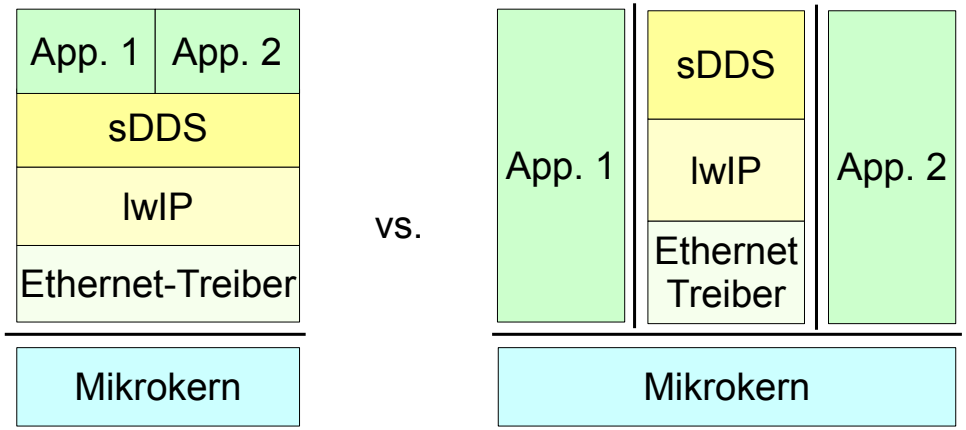
- Architektur
- DDS API Calls
- Senden
- Empfangen



Evaluation

Evaluation

Vergleich der Varianten a. und d.:



- Messung Performance-Overhead
  - Trace-Punkte an Komponentenübergängen toggeln GPIO-Pins
  - Aufzeichnung und Auswertung mit Logikanalysator
  - Ein Trace-Punkt kostet 0,920 µs
- Messung Speicherverbrauch RAM und Flash

Evaluation

Trace-Punkte *Sendevorgang* [in µs]

Phase	ohne Isolation				mit Isolation			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Aufruf DataWriter_write(); bei Isolation: RPC an sDDS					6,630	8,272	8,300	0,168
Empfang der RPC in sDDS					1,180	1,390	18,070	1,093
Ausführung DataWriter_write()	6,010	13,909	14,030	0,864	8,910	13,957	14,000	0,392
Generiere SNPS-Paket	4,090	5,482	18,060	0,736	4,130	5,409	6,230	0,103
sDDS UDP-Modul	7,130	7,145	7,390	0,020	7,130	7,142	7,390	0,020
lwIP Stack	23,140	23,175	23,570	0,031	23,210	23,240	23,610	0,030
Ethernet-Treiber	25,550	25,567	25,580	0,005	16,270	16,284	16,300	0,005
Zurück in Applikation					13,170	13,178	13,190	0,004

Gesamtzeit Senden	**75,3**	**88,9**
Bereinigt ohne Trace-Punkte	**70**	**81 (+16%)**

Evaluation

Trace-Punkte *Empfangsvorgang* [in µs]

Phase	ohne Isolation				mit Isolation			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Ethernet IRQ im Kern	6,040	6,762	12,120	0,074	6,010	6,707	12,140	0,097
Ethernet ISR im Ethernet-Treiber	8,240	8,276	18,380	0,428	8,240	8,291	18,380	0,573
lwIP Stack	29,310	36,457	41,890	0,178	31,220	36,501	43,670	0,195
sDDS UDP-Modul	17,490	17,505	23,380	0,125	17,410	17,426	20,320	0,073
sDDS DataSink_processFrame()	8,080	9,001	25,710	1,194	7,750	8,680	25,580	1,204
Data available Callback	5,630	5,639	10,940	0,081	5,460	5,471	8,490	0,054
sDDS sendet Event an CB-Manager					1,140	1,147	3,070	0,023
Aktivierung Callback Manager					20,380	20,415	27,990	0,367
Aufruf DataReader_take_next_sample(); Bei Isolation: RPC an sDDS					1,160	1,167	4,490	0,044
Empfang der RPC in sDDS					8,240	8,252	13,750	0,107
sDDS sendet RPC Reply					3,390	3,396	6,960	0,048
Daten in Applikation verfügbar	3,340	3,352	8,820	0,075	1,160	1,167	4,490	0,044

Gesamtzeit Empfangen **87,0** **127,0**

Bereinigt ohne Trace-Punkte **80** **115 (+44%)**

Evaluation

Speicherverbrauch

- RAM-Verbrauch: 60.592 Bytes → 65.792 Bytes
  - Anstieg: 8,7%
  - → Speicher für Stacks der hinzugefügten Tasks
- Flash-Verbrauch: 83.944 Bytes → 90.608 Bytes
  - Anstieg: 7,9%
  - → Zusätzlicher Code für Stubs, Tasks, Partitionen

Fazit

Fazit

- Zusätzliche Kontextwechsel sind teuer
  - Zusätzliche Tasks
  - Laden-/Umschalten der MPU-Konfiguration
- Nur moderater Anstieg im Speicherverbrauch
- Vergleichbare Arbeiten bisher Systeme mit MMU
- Der Ansatz bietet noch Optimierungspotential!

Vielen Dank

für Ihre Aufmerksamkeit!

Fragen?